Design of a CDD-Based Fault Injection Framework for AUTOSAR Systems

As'Ad Salkham, Antonio Pecchia, Nuno Silva

Design of a CDD-Based Fault Injection Framework for AUTOSAR Systems

As'ad Salkham[1], Antonio Pecchia[2] and Nuno Silva[1]

[1]ASD-T Aeronautics, Space, Defense and Transportation – Critical Software S.A.
Parque Industrial de Taveiro, Lote 49, 3045-504, Coimbra, Portugal

{asad.salkham, nsilva}@criticalsoftware.com

[2] Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione (DIETI)
Università degli Studi di Napoli Federico II
Via Claudio 21, 80125, Naples, Italy

antonio.pecchia@unina.it

Abstract. Over the past years, developing automotive software has been of an Electronic Control Unit (ECU)-specific nature despite the wide range of in-vehicle electronics. With the increasing maintainability cost of such an approach, the AUTomotive Open System Architecture (AUTOSAR) has emerged as a collective effort among different elements in the automotive industry in order to provide standardized and open software architecture for different types of vehicles. This paper presents a framework design to assess AUTOSAR systems by means of fault injection, which is recommended by the ISO 26262 standard for validating safety requirements at software, system and hardware level. Our proposal stems from a number of technical challenges characterizing AUTOSAR systems, and leverages AUTOSAR's Complex Device Driver (CDD) cross-layer and memory partitioning to support the implementation of a minimally intrusive fault injection framework. The potential of the approach in triggering error handling mechanisms implemented across the different layers of a given AUTOSAR system is discussed by means of examples.

Keywords: Fault injection, error handling, AUTOSAR, ISO 26262, validation.

1 Introduction

Vehicles are part of our daily life whether we use them privately or in public transportation. Despite the growing environmental concern, vehicles are continuously increasing in numbers per capita in many countries [1]. The reliance on using more and more ECUs in the automotive industry is clear. For example, some vehicles contain up to 70 ECUs [2]. Consequently, given the safety-critical nature of most vehicles, special attention is given to the development of those ECUs and to their affiliated cost particularly that electronics and software can amount to 40% of a given vehicle's overall cost [2]. Indeed, situations where vehicles have to be recalled due to system or software issues, (e.g., Honda issued a recall for ~2.5 million CRV and Accord sedan in 2011 due to a transmission software glitch [3]) have to be avoided.

Traditionally, the automotive sector has focused on developing software in an ECU-specific manner. This practice proved to be costly in terms of maintenance, lack of reusability, management and update. Consequently, a wide group of automotive manufacturers and third-party companies put in place a collective effort to realize a standardized and open software architecture that can be used for the diverse in-vehicle systems without compromising the quality and with cost-efficiency, i.e., **AUTOSAR** (www.autosar.org). The aim is to make automotive software development more of an application-specific nature. More important, AUTOSAR also faces the safety-critical nature of vehicle systems. It has to abide by high functional safety requirements during the development process and indeed in providing a set of recommended safety mechanisms. The ISO 26262 (tailored from IEC 61508) standard [4] provides the needed guidelines throughout the development process of automotive systems in order to achieve the required level of functional safety. ISO 26262 in its software (part 6), system (part 4) and hardware (part 5) product development guidelines recommends the use of **fault injection** as a means for testing automotive systems. On the software level, fault injection is essential for unit and integration testing given that the goal is an Automotive Software Integrity Levels (ASIL) C or D. The standard suggests to inject arbitrary faults to test the implemented safety mechanisms by corrupting values/variables, software or hardware. However, there is a lack of details concerning the required fault injection approach according to ISO 26262.

An AUTOSAR system (see **Fig. 1**) has a layered architecture involving heterogeneous components. Our fault injection approach (reflected in the framework design) aims to assess the safety and error-handling mechanisms distributed across *all* the layers of a given AUTOSAR system. The approach is motivated by a number of facts and limitations discussed onwards. Essentially, while commercially available AUTOSAR basic software implementations are certified and ISO 26262 complaint, third-party hardware and application software might have not gone through the same rigorous and extensive non-simulation-based validation activities. Also, AUTOSAR was built without taking explicitly fault injection needs into account, which resulted in the lack of required accessibility to either hardware or software interfaces in order to support the injection of faults. The later challenge is increased by the inability to alter proprietary certified AUTOSAR implementations to allow for fault injection. Moreover, given the distributed nature of error handling mechanisms in AUTOSAR, it is essential to test the system as a whole including the application software.

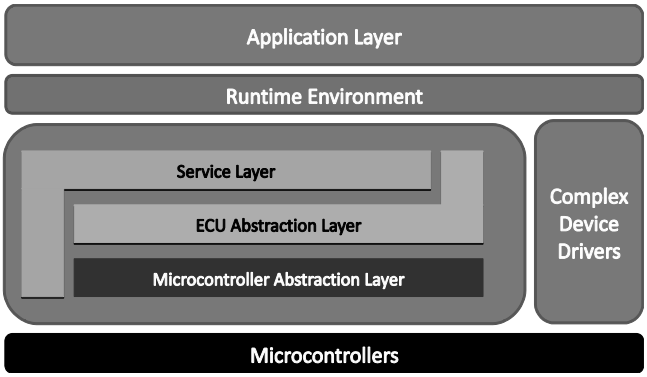

Fig. 1. AUTOSAR system architecture.

The proposed framework design does not assume the ability to modify AUTOSAR's basic software but rather benefits from the Complex Device Driver cross-layer (given its privileges) and memory partitioning to support the implementation of a minimally intrusive fault injection framework. The potential of the approach at assessing an AUTOSAR system at different layers is discussed by means of examples.

The remainder of this paper is organized as follows. Section 2 presents relevant background information while Section 3 describes AUTOSAR's safety mechanisms as well as the error model and the error handling in place. The design of our fault injection framework and the example scenarios are presented in Section 4. Section 5 provides a brief discussion and the possibility of future work, while Section 6 concludes the work.

2 Background and Related Work

This Section describes some essential concepts concerning AUTOSAR, fault injection, ISO 26262 and also presents related work.

2.1 AUTOSAR

AUTOSAR brings in the realization of an application-specific approach for automotive software development as opposed to an ECU-specific one. AUTOSAR provides a means for developing applications that are platform independent as long as they abide by a specified process and the interfaces provided. This helps in alleviating the burden of managing the complexity of the diverse network of electronics in vehicles without compromising quality and in a cost-efficient manner. It also provides for better maintainability through easing the exchange and update of software. The AUTOSAR architecture (see **Fig. 1**) mainly encompasses an application layer (comprising Software Components (SWC), a Run-Time Environment (RTE) and the Basic Software (BSW). Indeed, the underlying ECU is composed of a set of microcontrollers (µCs). The BSW comprises three main layers and a cross-layer; the services layer, the ECU abstraction layer, the microcontroller abstraction layer and the Complex Device Drivers (CDD) cross-layer. Furthermore, the CDD cross-layer provides a means to implement non-standardized functionalities with direct access to both the RTE and to the µCs. For example, in situations where complex or new sensors (or any unsupported hardware) are added, their specific drivers can be implemented in the CDD cross-layer benefiting from the fact that it mainly serves as a loosely coupled container. Indeed, the latter's software must abide by AUTOSAR's port and interface specifications.

2.2 Fault Injection

Fault injection is a widely adopted technique in the area of dependability evaluation that includes deliberate introduction of faults in a system with the aim of assessing its behaviour and evaluating the effectiveness of its fault-tolerance mechanisms [5]. The importance of fault injection is well recognised in the critical systems industry as such a technique allows for validating error-handling by triggering code paths that might be hard to test under regular operations. More important, fault injection is currently recommended by many international standards (e.g., ISO-26262, IEC-61508 and NASA-

STD-8719.13B) to support the system validation and certification process and to develop robust software.

A variety of fault injection approaches were proposed over the years. Early work in the area of fault injection aimed at emulating hardware problems by injecting physical faults (e.g., by means of radiation, shorting connections on circuit boards and interacting with the hardware device at pin-level) in the system [6]. These techniques became more and more unfeasible because of the growing hardware complexity, despite the specialized hardware developed to support their usage. For these reasons, Software-Implemented Fault Injection (SWIFI) has gained popularity.

SWIFI approaches, which are closer to the one proposed by our work, allow for emulating the effects of hardware faults through software. Emulation can be accomplished, for instance, by modifying the content of memory locations and registers with a corrupted value; corruption can affect both the program code and the program state. The adopted fault model often consists of bit-flip and stuck-at operators as they are accepted to be representative of hardware faults. SWIFI overcomes the limitations of physical fault injection and ensures a controlled framework to perform experiments. These features make SWIFI suitable within the critical industry domain [7]. Examples of SWIFI tools are Xception [8] and NFTAPE [9].

2.3 ISO 26262

The growth concerning safety-critical applications in the automotive market is a key concern to system engineers as they tackle safety challenges. The automotive industry will continue to improve on vehicle safety systems, for example, basic airbag-deployment systems, steering and braking systems, powertrain and body controller application, and Advanced Driver Assistance Systems (ADAS) with accident prediction and avoidance capabilities. These safety functions are increasingly implemented in electronics and this is where the ISO 26262 standard comes in to ensure the safety of these electronic systems by providing guidelines to help prevent severe failures and control them in case of their occurrence. ISO 26262 highly recommends fault injection for achieving ASIL C and D solutions.

2.4 Related Work

Currently, an increasing number of AUTOSAR BSW implementations, development tools and diagnostic environments are being adopted. Commercially, there are a number of companies that provide rather comprehensive AUTOSAR implementations, notably; Elektrobit and Vector. The latter provides MICROSAR and MICROSAR Safe that are both BSW implementations (AUTOSAR v4.0) while the latter is meant to comply with ISO 26262 up to ASIL D. Vector also provides CANoe which is a software tool that can be used for the development, testing and analysis of individual ECUs and networked ones with a special support for development in an AUTOSAR environment, i.e., CANoe AUTOSAR Monitoring and Debugging (CANoe.AMD). Another Vector tool is DaVinci; a SWCs developer and tester that also has an RTE configurator. Moreover, Elektrobit provides a set of comprehensive solutions (EB Tresos) to allow for developing AUTOSAR v4.0 compliant ECU software (up to ASIL D) in addition to supporting debugging and tracing. Elektrobit's Tresos AutoCore provides a BSW with an operating system that supports single and multicore processing.

Other AUTOSAR commercial solution providers include Kpit Cummins, Mecel & Mentor Graphics, Bosch (CUBAS & iSolar IDE), etc. On the open source front, Arctic provides development tools and a BSW implementation under GNU GPL.

Concerning work related to fault injection particularly for AUTOSAR, only a limited number exist. In [2] a SWIFI framework for AUTOSAR is presented. They exploit the CANoe simulation environment to introduce errors through the so called suppression and manipulation hooks. These hooks are inserted in selected AUTOSAR API implementations to either return a specific error code (suppression) or corrupt certain data structures (manipulation). However, similar to other simulation-based approaches and as acknowledged, the approach is restricted by the simulation environment accessibility issue where the injection has to abide by the level of abstraction provided by the CANoe let alone the probe effects of the simulation itself. Moreover, in [10], fault injection can be seen as a profound choice for assessing the coverage of the proposed fault-tolerance scheme, i.e., "defence software" within an automotive embedded software context. Notably, in [11] an instrumentation framework for assessing the dependability of AUTOSAR systems at the software component level is presented. The framework supports instrumentation at source code, header files and object code levels. Promising results were presented in the evaluation in terms of overhead in source lines of code and execution time for a typical anti-lock braking system as well as in a fault injection experiment. Some limitations were also discussed besides what was described as conceivable workarounds. However, the fault injection experiment constituted bit flipping with no relation to any AUTOSAR fault model.

3 Safety Mechanisms in AUTOSAR

AUTOSAR includes a variety of mechanisms aiming to mitigate the severity of runtime errors and to achieve safety requirements. Some safety mechanisms, e.g., watchdogs, redundancy and memory partitioning, are directly supported by the BSW to achieve ISO 26262 compliancy; nevertheless, other mechanisms such as error handling support (described in Section 3.2), can be implemented either at the BSW or at the SWC layer. Consequently, evaluating the behaviour of AUTOSAR systems as a whole is crucial. The main concepts related to the error model and to the error handling mechanisms introduced by AUTOSAR are briefly described in the following.

3.1 Error Model

An error is the part of the total system state that may lead to its subsequent failure, i.e., a deviation from the correct system service [12]. Errors are caused by faults (e.g., hardware problems, bugs in the source code of the program or misconfigurations) that are activated during system operations. AUTOSAR makes extensive use of error models to support the specification of its error handling mechanisms. Error models aim to cover the different ways errors manifest as a consequence of fault activations. AUTOSAR standard errors include errors that are mostly handled by the BSW. Only two categories of standard errors are currently provided by AUTOSAR, i.e., communication-related errors (including Controller Area Network (CAN)-related errors) and Non-Volatile Random Access Memory (NVRAM)-related errors. Examples of CAN-related errors are bus off and transmission buffer full, while examples of NVRAM-

related errors are Cyclic Redundancy Check (CRC) error and internal flash write job error. AUTOSAR encompasses five error categories:

- Data: erroneous value of a function parameter, variable or message;
- Program flow: the flow of the program is different than the expected one (e.g., missing or wrong instructions are executed by the software);
- Access: a component tries to access a resource (e.g., a memory partition) without the proper access rights;
- Timing: an operation (e.g., packet transmissions or required functions) is delivered early, late or it is completely omitted;
- Asymmetric: Byzantine behaviour; no assumptions can be made about the erroneous component.

For example, a bug in the source code of the program (e.g., a faulty instruction such as a wrong condition within an "if statement") might result into a data error or a program flow error. Also, other type of errors is the development errors; meaning that they are supposed to have been handled before deployment, e.g., NULL pointer calls.

3.2 Error Handling

Error handling is concerned with the treatment of errors occurring during the system operation, i.e., in terms of detection, isolation, and recovery. According to the AUTOSAR specification, error handling can be implemented either at the BSW or at the SWC layer; furthermore, SWCs could be expected to handle some error notifications coming from the BSW. AUTOSAR provides a detailed specification (including information flow) of error handling mechanisms addressing standard errors. Not all of the standard errors are fully handled within the BSW. For example, the COM TX deadline monitoring, which represents a communication error, is detected at the BSW; however, an SWC is expected to recover from the error (e.g., by re-transmitting the signal).

AUTOSAR error handling exploits a variety of techniques (e.g., plausibility checks, agreement, checksums/codes, execution sequence monitoring and reset) that represent well-established and consolidated means to develop dependable software. Each mechanism addresses one (or more) error categories(s) and can partially or fully accomplish error treatment tasks. For example, checksums, which involve calculating extra information for a given data, target the mentioned data errors and allows for detecting and, under certain hypothesis, for restoring the original value of the corrupted data. Similarly, execution sequence monitoring focuses on control paths taken by the program and it is able to detect program flow errors; however, the treatment of the error should involve other mechanisms, e.g., reset, to accomplish recovery.

4 CDD-Based Fault Injection Framework

The CDD-based approach for fault injection stems from a number of motives. Fault injection is highly recommended by ISO 26262 for testing at different development stages. While the deployment of a fault injection approach normally requires specific support and often changing the code or interface parameters at varying levels, ISO 26262 certified AUTOSAR implementations are proprietary. For these reason, assess-

ing an AUTOSAR system, including third-party applications developed on top of it, becomes even more challenging. The challenge increases given the fact that error handling is distributed over different AUTOSAR layers.

Consequently, a minimally intrusive, (i.e., not requiring heavy access/change to AUTOSAR's BSW) and generic approach for assessing AUTOSAR systems using fault injection is needed. To this objective, we exploit the CDD cross-layer (which comes with the architecture to handle unsupported microcontrollers) and the memory partitioning feature. The key idea behind our approach is to trigger faults from the very base layer, i.e., µCs, and to monitor the AUTOSAR system's behaviour with the aim of evaluating its safety as a whole. More important, running the fault injection monitoring and control logic in a separated memory partition will ensure the lack of interference with the solution under testing.

In **Fig. 2**, the modules of the fault injection framework which represents our approach and highlights relationships with the AUTOSAR architecture are presented. The fault injection SWCs will mainly constitute a controller and a monitoring service as in a regular fault injection framework; injection is performed at the CDD level. The components are detailed in the following.

Fault injection modules, i.e., the entities of the framework that will introduce faults into the system, are implemented within the CDD cross-layer as it can interface selected µCs directly. Furthermore, a CDD has the privilege to communicate with SWCs through the RTE, which makes it a suitable candidate for placing the fault injection logic. Each Fault Injection (FI) module is specialized in emulating certain errors, e.g., communication-related, WatchDog Timer (WDT) or NVRAM-related: emulation is achieved by corrupting the status, behaviour or content of the µCs through the CDD. It must be observed that µCs are not our test target and that the corruption aims to activate the error handling and safety mechanisms at the BSW and/or at the SWCs that rely on the integrity of such µCs. For this reason, it is sufficient to have the needed µC for a specific fault-triggering scenario being accessible and adjustable.

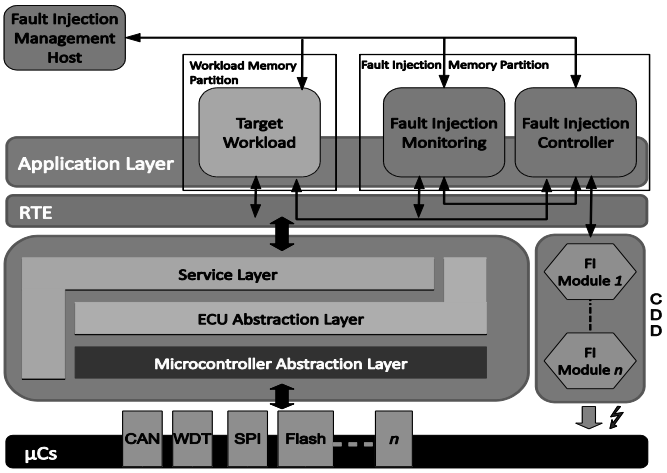

Fig. 2. CDD-Based Fault Injection

The framework consists of other entities, i.e., the target workload and the fault injection monitoring component. The target workload is implemented by one or more

SWCs that use the BSW in a way that can exercise error-prone activities such as memory access or communication. The monitoring SWC is responsible for gathering information concerning the system status. Typically, it will require polling, through the RTE, components that are responsible for receiving error reports or possibly those performing mitigation. For example, monitoring may involve the Diagnostic Event Manager (DEM), which is responsible for most of the housekeeping, CAN interface, workload SWC, NV memory manager.

The fault injection controller coordinates the described entities and is responsible for iterating through the fault injection experiments composing a campaign. It is worth noting that we exploit AUTOSAR memory partitioning to isolate the fault injection software from the system targeted by the injection. More in details, controlling and monitoring software components run on a separate partition from that of the workload. This ensures that the fault injection software will be isolated from any potential memory errors that could occur at the workload partition. The controller is able to activate/deactivate a given injection module at the CDD and run the workload through the RTE.

The fault injection management host is responsible for loading the target workload and more importantly receiving the results produced by the monitoring component during each experiment for subsequent analysis. Also, it is responsible for defining the fault injection campaigns to be passed on and carried out by the fault injection controller.

4.1 Example Scenarios

A pair of example scenarios is discussed here to demonstrate the potential use of the CDD-based fault injection framework in AUTOSAR systems. The fault injection management host is not presented in the scenarios but it should be assumed implicitly there. The examples show that injection experiments conducted at the very base layer of the system allow for triggering the error handling mechanisms implemented across the different layers. The first example is concerning a communication-related error while the second is about an NVRAM-related error. In both cases, the role of the components of the fault injection framework is discussed.

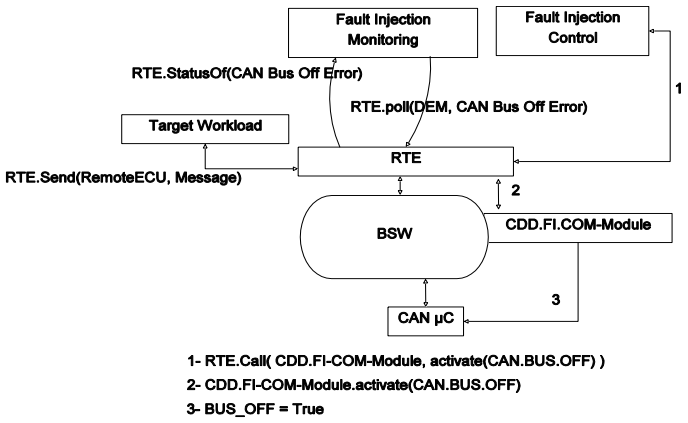

Fig. 3. Injection in the communication module.

Communication Error. The error targeted by the example is the CAN Bus Off error, which is raised when there is a CAN communication channel loss. In order to test for the CAN Bus Off error, a scenario is presented (see **Fig. 3**) where a target workload is trying to send a message to a remote ECU that eventually requires accessing the CAN bus. Following the target workload request, the CAN Bus Off error is triggered as the CAN µC is forced to be offline by the CDD fault injection module. The latter was instructed through the RTE to make the CAN bus inaccessible as part of the fault injection logic. The information path of this error is presented in **Fig. 4**. It can be observed that, despite injection is conducted at the hardware level, the error handling process is spread over different components at different levels. Each component has a specific role in the detection and recovery processes. In this case, the CAN driver is responsible for polling its dedicated µC to check the status of the CAN bus readiness, i.e., the communication channel aliveness. The CAN driver is able to discover the CAN Bus Off error as the µC updates its register where the driver then tries to cancel all pending messages and informs the CAN interface about the error using a dedicated API. The CAN interface hence changes its mode to "stopped" and reports the error to the CAN state manager using a similar dedicated API. The CAN state manager starts the recovery and counts the error events. If the error is confirmed, the DEM, the BSW state manger and the communication manager are notified. The recovery includes resetting the CAN µC and enabling/disabling the transmit path. If the error is successfully mitigated, its event is removed from the DEM and the CAN state manager informs the involved elements. Indeed, the relevant SWCs are always updated through the communication manager using the RTE about the detection and the recovery of the error at hand. As reported in **Fig. 3**, the fault injection monitoring SWC polls the DEM through the RTE in order to register a successful or a fail error handling for that case. Indeed, the fault injection control is responsible for the process order and ensures such an order through communication with the target workload SWC, the monitoring SWC and the CDD, all through the RTE. Moreover, the fault injection monitoring SWC considers that the error is successfully handled if its relevant event is no more present at the DEM.

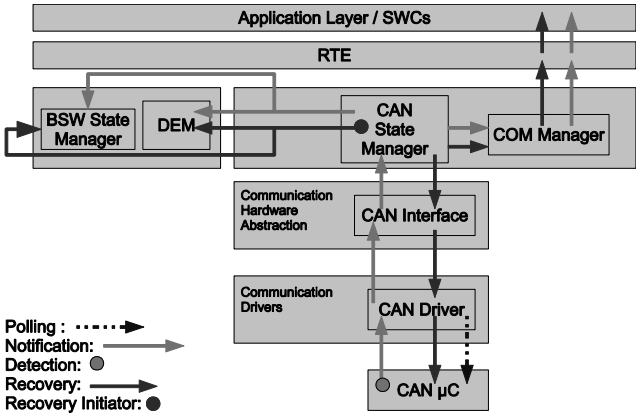

Fig. 4. CAN Bus Off Error: Information path.

NVRAM Error. The error targeted here is related to memory and specifically the CRC of memory blocks. In order to test for the CRC error, an instance of the fault injection framework is presented in **Fig. 5** where a target workload is trying to read a specific memory block. Similar to the scenario in the first example, the fault injection controller organizes the process order. To allow for a CRC error to be triggered, the controller requests that the specific memory block planned to be read by the target workload to have its CRC bits corrupted. This is done through a request to the dedicated fault injection CDD module, i.e., the CDD-FI-NVRAM-Module which in its turn reflects that on the requested block's CRC. The latter can be corrupted, for instance, by bit flipping all its bits using the xor operation with a stream of one bits.

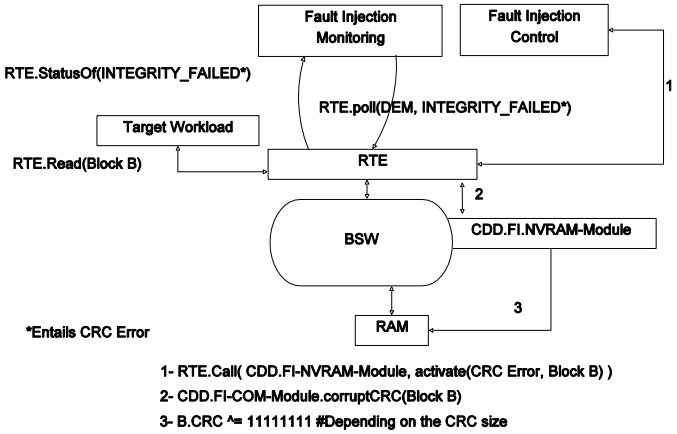

Fig. 5. Injection in the NVRAM module.

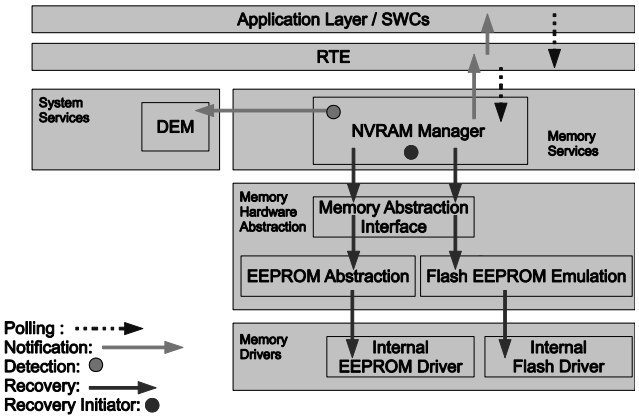

Fig. 6. CRC: Information path.

The information path of this error is presented in **Fig. 6**. The NVRAM manager verifies whether there is a CRC mismatch after a read operation on a given RAM block. If a mismatch exists, the DEM is informed of an integrity error and the recovery process is initiated. The recovery includes operations like a read retry, reading redundant block

and/or reading from Read-Only Memory (ROM) block. The recovery requests normally propagate from the NVRAM manager through the lower memory-related layers, i.e., the memory hardware abstraction and the memory drivers. If the recovery entails a loss or redundancy, the NVRAM also reports that to the DEM. The application layer shall poll this type of error and is able to be notified through the RTE. Again, error handling involves different layers of the AUTOSAR system. As the NVRAM manager detects the mismatch, the fault injection monitoring keeps polling the DEM to check the existence of an integrity error and/or loss of redundancy errors.

5 Discussion and Future Work

This work is meant to highlight the potential of using the CDD cross-layer and memory partitioning as a means for minimally intrusive fault injection in AUTOSAR. To our knowledge, this is a new approach that can exploit the privileges given to CDDs in terms of accessing the RTE and µCs directly. This allows for testing the safety mechanisms and error handling in a given AUTOSAR system as a whole and without the need to change the BSW's code. This is a clear advantage when dealing with systems like AUTOSAR where error handling is well-spread among different components in the BSW and sometimes in the SWCs. Also, the issue of accessibility in proprietary commercial solutions can be circumvented if a fault injection technique is presented in this manner. One concern that can be raised is the extent to which µCs can be altered if they are already certified to be protected. The answer to such a concern is that the framework is not meant to test µCs but rather use them as fault-triggering points and indeed any type of µCs can be used as long as they satisfy the planned alteration requirements set by the fault injection CDD. The approach carries a promising idea that we intend to provide a prototype for in the near future.

6 Conclusion

This paper presented a framework design that provides a minimally intrusive and generic approach for assessing AUTOSAR systems by means of fault injection. It highlighted the need for such an approach given the increasing safety requirements in the automotive industry and the need to abide by strict standards like ISO 26262 where fault injection is highly recommended for testing at different levels. In order to understand the safety issues in AUTOSAR, the architecture was described including the error model, safety and error handling mechanisms as well as the relation to ISO 26262. The CDD-based fault injection framework was detailed including a pair of examples concerning AUTOSAR's communication-related and memory-related errors. Very few work exist on fault injection in AUTOSAR systems, hence this work presents a novel approach that can help in realising an AUTOSAR system-wide assessment using minimally intrusive fault injection. We believe that the use of CDDs and memory partitioning for fault injection (in AUTOSAR systems) is a clear and sound approach that will prove to be essential in the near future.

Acknowledgements. This work has been supported by the European project CRITICAL Software Technology for an Evolutionary Partnership (CRITICAL STEP, www.critical-step.eu), Marie Curie Industry-Academia Partnerships and Pathways (IAPP) number 230672 in the context of EU's FP7, and by the project "EMBEDDED Systems" CUP B25B09000100007, POR Campania FSE 2007/2013.

References

1. U.N. Division for Sustainable Development, "Transport Report," United Nations, New York, 2010.
2. P. Lanigan, P. Narasimhan and T. Fuhrman, "Experiences with a CANoe-based fault injection framework for AUTOSAR," in Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on, Chicago, 2010.
3. http://www.insideline.com, "Honda Will Recall 2.5 Million Cars Worldwide To Fix Transmission Software," Insideline, 8 August 2011.
4. International Organization for Standardization. Product development: software level. ISO/DIS 26262-6, 2009.
5. J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications," IEEE Transactions on Software Engineering, vol. 16, no. 2, pp. 166-182, 1990.
6. U. Gunneflo, J. Karlsson and J. Torin, "Evaluation of error detection schemes using fault injection by heavy-ion radiation," in International Symposium on Fault Tolerant Computing (FTCS-19), 1989.
7. A. Pecchia, A. Lanzaro, A. Salkham, M. Cinque and N. Silva, "Leveraging Fault Injection Techniques in Critical Industrial Applications". In Innovative Technologies for Dependable OTS-Based Critical Systems, pp 131-141, Springer, 2013.
8. J. Carreira, H. Madeira and J. Silva, "Xception: a technique for the experimental evaluation of dependability in modern computers," IEEE Transactions on Software Engineering, vol. 24, no. 2, pp. 125-136, 1998.
9. D. Stott, B. Floering, D. Burke, Z. Kalbarczpk and R. Iyer, "NFTAPE: a framework for assessing dependability in distributed systems with lightweight fault injectors," in IEEE Computer Performance and Dependability Symposium, 2000.
10. C. Lu, J.-C. Fabre and M.-O. Killijian, "An approach for improving Fault-Tolerance in Automotive Modular Embedded Software," in 17th International Conference on Real-Time and Network Systems, Paris, 2009.
11. T. Piper, S. Winter, P. Manns, and N. Suri. 2012. Instrumenting AUTOSAR for dependability assessment: A guidance framework. In Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (DSN '12). IEEE Computer Society, Washington, DC, USA, 1-12.
12. A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. "Basic Concepts and Taxonomy of Dependable and Secure Computing". IEEE Transactions on Dependable and Secure Computing, 1:11-33, Jan 2004.